Corpus transformations
Praktikum Verarbeitung natürlicher Sprache

Richard Mörbitz

14.06.2021
Praktical problems in parsing

Our basic CFG parser has some issues:

- Only binary and unary rules can be processed, but the training corpus is \( n \)-ary

\[
\begin{align*}
A & \rightarrow BC \\
A & \rightarrow a
\end{align*}
\]

\( \text{vs.} \)

\[
\begin{align*}
A & \\
B & \\
C & \\
D & \\
e & \\
f & \\
g
\end{align*}
\]

- Training corpus does not contain all English words

He is a brilliant deipnososophist. \( \rightsquigarrow \) No parse!

- Sparse data problem: rare words in the training corpus may get atypical probabilities
Outline

1. Binarization and Markovization
2. Unknown/rare word handling
Binarizing the training corpus vs. binarizing the grammar
Straight-forward binarization

Binarizing the training corpus vs. binarizing the grammar
Straight-forward binarization

Binarizing the training corpus vs. binarizing the grammar

Each occurrence of two consecutive nonterminals yields a new nonterminal

No information about context is preserved

Number of right-hand side nonterminals is fixed

\[ S \rightarrow A \ A \ A \ \rightarrow \ A \ X_1 \ \ (1) \]
Straight-forward binarization

Binarizing the training corpus vs. binarizing the grammar

Problems:
1. Each occurrence of two consecutive nonterminals yields a new nonterminal $X_i$
Straight-forward binarization

Binarizing the training corpus vs. binarizing the grammar

\[ S \rightarrow \langle A, A \rangle \]

Problems:

1. Each occurrence of two consecutive nonterminals yields a new nonterminal \( X_i \).
Binarizing the training corpus vs. binarizing the grammar

Problems:
1. Each occurrence of two consecutive nonterminals yields a new nonterminal $X_i$
2. No information about context is preserved
Straight-forward binarization

Binarizing the training corpus vs. binarizing the grammar

$$S \quad \rightarrow \quad S|\langle A, A \rangle \quad (3)$$

Problems:

1. Each occurrence of two consecutive nonterminals yields a new nonterminal $X_i$
2. No information about context is preserved
Straight-forward binarization

Binarizing the training corpus vs. binarizing the grammar

![Binary tree diagram]

Problems:
1. Each occurrence of two consecutive nonterminals yields a new nonterminal $X_i$
2. No information about context is preserved
3. Number of right-hand side nonterminals is fixed
Vertical and horizontal Markovization

Previous slide  Markovization with $v = 1$ and „$h = \infty$“

In general  store $v \in \mathbb{N}_{+}$ nonterminals towards the root and
$h \in \mathbb{N}_{+}$ nonterminals to the left

1:  
   \begin{align*}
   \text{function} & \quad \text{MARKOVIZE}(t = \sigma(t_1, \ldots, t_k)) \\
   2: & \quad \text{if} \ t \ \text{is preterminal} \ \text{then} \\
   3: & \quad \quad \text{return} \ t \\
   4: & \quad \text{else if} \ k \leq 2 \ \text{then} \\
   5: & \quad \quad \text{return} \ \text{ADD_PARENTS}(\sigma)(\text{MARKOVIZE}(t_1), \ldots, \text{MARKOVIZE}(t_k)) \\
   6: & \quad \text{else} \\
   7: & \quad \quad \sigma' \gets \text{ORIGINALLABEL}(\sigma)|\langle \text{label of } t_2, \ldots, \text{label of } t_{h+1} \rangle \\
   8: & \quad \quad \text{return} \ \text{ADD_PARENTS}(\sigma)(\text{MARKOVIZE}(t_1), \text{MARKOVIZE}(\sigma'(t_2, \ldots, t_k)))
   \end{align*}

Note:
- \(\text{ADD_PARENTS}(\sigma) = \sigma^\langle l_1, \ldots, l_{v-1} \rangle\), where the $l_i$ are the labels of the ancestors of $\sigma$ which occur in the original tree
- If $v = 1$ or there are no parents, leave out $^\langle \rangle$
Markovization – example

here: $v = 3, h = \infty$

markovize → ROOT

FRAG

RB
Not
this

NP-TMP

DT
NN
year
Markovization – example

here: $v = 3$, $h = \infty$
Markovization – example

here: $v = 3, h = \infty$

markovize

ROOT

FRAG $^<\text{ROOT}>$

RB

FRAG $|<\text{NP-TMP},.>$

Not

NP-TMP

DT

NN

this

year
Markovization – example

here: $v = 3$, $h = \infty$
Markovization – example

here: \( v = 3, h = \infty \)
Markovization – example

here: $v = 3, h = \infty$
Markovization – example

here: $v = 3$, $h = \infty$
Markovization – example

here: $v = 3$, $h = \infty$
Markovization – example

here: $v = 3$, $h = \infty$
Markovization – example

here: \( v = 3, h = \infty \)
Markovization – example

here: $v = 3, h = \infty$

markovize

ROOT

FRAG^<ROOT>

RB

Not

NP-TMP^<FRAG,ROOT>

DT

NN

this

year
Markovization – example

here: $v = 3$, $h = \infty$

markovize

```
markovize
     /
    /  \
ROOT /    \
   /     \
FRAG<ROOT>
   /     \
RB  /     \
   /      \
Not/       \
   /        \
NP-TMP<FRAG,ROOT>
   .
   /
DT    \
   /
this

   /
NN    \
   /
year
```

Richard Mörbitz: Corpus transformations (Praktikum Verarbeitung natürlicher Sprache) 14.06.2021 6 / 14
After parsing: debinarization

Why?
- Debinarized parse trees are linguistically relevant
- Comparison with (debinarized) gold corpus

1: function DEBINARIZE(t = σ(t₁, ..., tₖ))
2: if ROOT(tₖ) is Markovization node with children t'₁, t'₂ then
3:   return DEBINARIZE(σ(t₁, ..., tₖ₋₁, t'₁, t'₂))
4: else
5:   return σ(DEBINARIZE(t₁), ..., DEBINARIZE(tₖ))

Also: remove ancestor annotation from each node (not shown)!

In material: both binarized and (debinarized) gold corpus
Debinarization – example

debinarize → ROOT

FRAG^<ROOT>

RB

FRAG|<NP-TMP,>.^<ROOT>

Not

NP-TMP^<FRAG,ROOT>

DT

NN

this

year
Debinarization – example

if cond. is true ⇒ call in l. 3

debinarize
Debinarization – example

(debinarize)

ROOT

FRAG

NP-TMP^<FRAG,ROOT>

RB

Not

debinarize

RB

this

DT

year

NN
Debinarization – example

```
not this year.
```

```
debinarize
  └── ROOT
      └── FRAG
          └── NP-TMP^<FRAG,ROOT>
              └── DT
                  └── NN
                      └── this
```

```
Debinarization – example

debinarize

ROOT

FRAG

NP-TMP^<FRAG,ROOT>

RB
Not

DT
this

NN
year
Debinarization – example

Debinarize

ROOT

FRAG

NP-TMP<FRAG,ROOT>

RB

Not

DT

NN

this

year
Debinarization – example

```
    debinarize
   /        \     
  /          \    
 ROOT       FRAG
 /           \  
RB          NP-TMP
 /  \      /  \  
Not  DT  NN
   /    /    |
  this year
```

Debinarization – example

```
rb  frag
  not  np-temp
        nn  nn
        this year
```
Debinarization – example

debinarize

ROOT

FRAG

RB
Not

NP-TMP

DT

NN

this

year
Debinarization – example

debinarize

ROOT

FRAG

RB

NP-TMP

Not

DT

NN

this

year
Debinarization – example

debinarize

ROOT

FRAG

RB
Not

NP-TMP

DT
this

NN
year
Outline

1. Binarization and Markovization

2. Unknown/rare word handling
Basic unking

Idea:

1. Replace words that are *unknown* to the parser by an „unknown token“ (here: UNK)
2. Assign some probability mass to rules that generate UNK
Basic unking

**Idea:**
1. Replace words that are *unknown* to the parser by an „unknown token“ (here: UNK)
2. Assign some probability mass to rules that generate UNK

**Implementation:**
1. Reflection in the parser: before parsing $w$ (where $\Sigma$ is the set of terminals)
   
   ```
   1: wmap ← \{(w_i \mid i \in \{1, \ldots, |w|\})
   2: for i = 1, \ldots, |w| do
   3: \quad if w_i \not\in \Sigma then
   4: \quad \quad w_i ← UNK
   ```

   and after parsing $w$ as $t$, restore original words

   ```
   1: for i = 1, \ldots, |w| do
   2: \quad LEAVES(t)[i] ← wmap[i]
   ```
Reflection in the grammar: replace rare words in the training corpus by UNK

Require: tree corpus corpus with terminal alphabet \( \Sigma \), threshold \( \in \mathbb{N}_+ \)
Ensure: every word occurring \( \leq \) threshold times in corpus is replaced by UNK

1: \( \text{wordcount} \leftarrow (0 \mid i \in \Sigma) \)
2: \textbf{for } \( t \in \text{corpus} \) \textbf{do}
3: \hspace{1em} \textbf{for } \( i = 1, \ldots, |\text{LEAVES}(t)| \) \textbf{do}
4: \hspace{2em} \text{wordcount[LEAVES}(t)[i]] \leftarrow \text{wordcount[LEAVES}(t)[i]] + 1
5: \textbf{for } \( t \in \text{corpus} \) \textbf{do}
6: \hspace{1em} \textbf{for } \( i = 1, \ldots, |\text{LEAVES}(t)| \) \textbf{do}
7: \hspace{2em} \textbf{if } \text{wordcount[LEAVES}(t)[i]] \leq \text{threshold} \textbf{ then}
8: \hspace{3em} \text{LEAVES}(t)[i] \leftarrow \text{UNK}

The grammar is induced from the modified corpus!
**Observation:** each unknown word is assigned the same probability
- bad language model: certain words are more likely to occur
- may worsen the parsing of sentences with rare words

**Solution:** categorize unknown words based on their *signature* [1]

\[
    w_i \leftarrow \text{UNK} \quad \Rightarrow \quad w_i \leftarrow \text{getSignature}(w_i, i)
\]

\[
    \text{LEAVES}(t)[i] \leftarrow \text{UNK} \quad \Rightarrow \quad \text{LEAVES}(t)[i] \leftarrow \text{getSignature}(w_i, i)
\]

- Some signatures can be found in the source code of the Berkeley parser
- Next slide: „unknownLevel = 4“
- Heuristics, prone to overfitting
- Be creative, but don’t overdo it!
1: function getSignature(word, i)
2:    if |word| = 0 then return UNK
3:    letterSuffix ← ISUPPER(word) ∧ NONE(ISLOWER, word) ⇒ -AC
              ISUPPER(word) ∧ i = 1 ⇒ -SC
              ISUPPER(word) ⇒ -C
              ANY(ISLOWER, word) ⇒ -L
              ANY(ISLETTER, word) ⇒ -U
              otherwise ⇒ -S
4:    numberSuffix ← ALL(ISDIGIT, word) ⇒ -N
              ANY(ISDIGIT, word) ⇒ -n
              otherwise ⇒ ε
5:    dashSuffix ← ANY(= '‐', word) ⇒ -H
              otherwise ⇒ ε
6:    periodSuffix ← ANY(= '.', word) ⇒ -P
              otherwise ⇒ ε
7:    commaSuffix ← ANY(= ',', word) ⇒ -C
              otherwise ⇒ ε
8:    wordSuffix ← |word| > 3 ∧ ISLETTER(word) ⇒ TOLOWER(|word|)
              otherwise ⇒ ε
9:    return UNK · letterSuffix · numberSuffix · dashSuffix · periodSuffix · commaSuffix · wordSuffix
What to do?

Until 14.07.2020, 23:59,

- implement debinarization (3a)
- implement trivial unkinking for corpus and adapt parser (3b)
- implement 3 of
  - Markovization
  - smoothing
  - pruning
  - $n$-best parsing
  - heuristic search

} optimizations, next week

All tasks' solutions are one submission!
You may send in your solutions earlier for feedback.
References I